NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

RANDOMIZATION TESTING OF MACHINE INDUCED RULES

by

Eric Dean Berry

September, 1995

Thesis Advisor: Balasubramaniam Ramesh

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE Randomization Testing of Machine Induced Rules			5. FUNDING NUMBERS
6. AUTHOR(S) Berry, Eric D.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Department of Defense (DOD) possesses tremendous amounts of data stored in many large databases. Given the size of these databases, large scale data analysis tools are required to find previously unknown and interesting patterns. Data Mining tools which produce output in the form of production rules, i.e., "If x, Then y" are preferred because the generated rules are understandable by humans and readily support decision making processes.

This thesis investigates the problems associated with the statistical testing of rule generated from data mining systems. Statistical testing of rules generated by data mining systems is required to ensure that the generated rules are based on valid statistical relationships and are not the result of random variation in the underlying data. A strategy for the testing of rules using a non-parametric test known as the randomization test is implemented for the testing of rules from a prototype data mining system.

14. SUBJECT TERMS Data Mining, Randomization Testing, Machine Learning			15. NUMBER OF PAGES 90
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

RANDOMIZATION TESTING OF MACHINE INDUCED RULES

Eric D. Berry
Lieutenant Commander, United States Navy
B.A., Louisiana State University, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Author: [illegible]
Eric D. Berry

Approved by: [illegible]
Balasubramaniam Ramesh, Thesis Advisor

William J. Haga
William J. Haga, Thesis Associate Advisor

Reuben T Harris
Reuben T. Harris, Chairman, Department of Systems Management

ABSTRACT

The Department of Defense (DOD) possesses tremendous amounts of data stored in many large databases. Given the size of these databases, large scale data analysis tools are required to find previously unknown and interesting patterns. Data Mining tools which produce output in the form of production rules, i.e., "If x, Then y" are preferred because the generated rules are understandable by humans and readily support decision making processes.

This thesis investigates the problems associated with the statistical testing of rule generated from data mining systems. Statistical testing of rules generated by data mining systems is required to ensure that the generated rules are based on valid statistical relationships and are not the result of random variation in the underlying data. A strategy for the testing of rules using a non-parametric test known as the randomization test is implemented for the testing of rules from a prototype data mining system.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Private and Government organizations commonly collect information pertaining to the operation of the organization. The amount of information collected has increased in recent years due to automation and of many routine business practices and the decreasing cost of storage media for holding the information. Banks, retailers, scientific and engineering research organizations, and government agencies all maintain large databases of information collected in their daily operations.

Clearly the means available to store information has increased. Organizations may collect gigabytes of information daily and have databases on an organizational scale in the terabyte range. However, many experts perceive that there is a gap between data generation and data understanding. [1] This growing gap between the large amount of data being stored and the ability to analyze and make use of this data has created interest in large scale data analysis tools. These tools are designed to search the organizations' databases for new and interesting patterns which were previously unknown and will be of future benefit to the organization. Applications for this technology exist in marketing, scientific research, and for development of expert systems. "Data Mining" tools are now becoming available to assist in large scale data analysis.

Data mining tools typically represent patterns in the database in the form of production rules, i.e., If x, then y. The rules produced by the data mining tool also must have a measure of certainty associated with the pattern which corresponds to the number of incorrect classifications in proportion to the number of correct classifications. Statistical hypothesis testing is customarily employed to test the validity of these rules to ensure that the induced rule is based on statistically valid relationship and is not the result of random variation in the underlying data.

B. RESEARCH OBJECTIVE

The primary objective of this thesis is examine the issues and problems associated with statistical hypothesis testing of rules induced by data mining tools. The problems of statistical hypothesis testing of rules induced by data mining tools by conventional testing methods are revealed and an implementation of an alternative strategy based on a non-parametric testing method known as randomization testing is implemented. Randomization testing methods are used to test the statistical significance of a set of rules produced by the Naval Postgraduate School Genetic Program (NPSGP) which is a prototype data mining system. The set of rules produced from NPSGP are tested to see if the rules produced from NPSGP are, in fact, better than those which would be expected to develop as a result of random variations in the underlying data.

C. ORGANIZATION OF THE STUDY

Chapter I provides the general introduction to the subject. Chapter II provides background on the process of data mining and principles behind the genetic programming methods utilized in NPSGP. Chapter III provides information associated with statistical testing of rules induced by data mining tools and presents the theory underlying randomization testing. Chapter IV provides information on the data and methodology used in the study and Chapter V provides results of the statistical testing. Chapter VI contains conclusions and recommendations of the thesis.

II. BACKGROUND

A. PRINCIPLES OF KNOWLEDGE DISCOVERY

1. Deductive Inference versus Inductive Inference

Deductive inference and inductive inference are logical thought processes used by humans to learn facts, theories, and principles. Although their goal is the same, to learn about the world around us, they differ in the manner in which conclusions are arrived at.

a. Deductive Inference

Deductive inference is the development of some theory or axiom which would then be proven or disproved by supporting or contradicting facts. For example, a theory such as 'February is the coldest month of the year' would be proven or disproved according to the empirical evidence available. Deductive inference in the most basic form contains the following steps:

1. Development of some preliminary hypothesis or theory
2. Collection of observational statements of facts
3. Judgement on quality of a hypothesis based on collected facts and initial premises

b. Inductive Inference

Inductive inference is the development of some premise or theory based on existing knowledge. Given a database of previous year's monthly temperature readings, a researcher would then reach his or her own conclusion which month is the coldest of the year. Inductive inference thus attempts to develop a theory or model of the real world based on available evidence to support the theory.

Inductive inference in its most basic form contains the following steps:

1. Collect observational statements of facts
2. Develop theory or model based on known facts

2. Inductive Learning

Learning is defined as "... changes in the system that are adaptive in the sense that they enable the system to do some task or tasks drawn from the same population more efficiently and more effectively the next time. [2] Learning is desirable because we seek improved ways to coexist with our environment to improve our welfare. Inductive learning is the application of inductive inferences to the process of learning.

Two major types of inductive learning are learning from examples (concept acquisition) and learning from observation (descriptive generalization). [3] Learning from examples involves placing objects into logical conceptual categories such as weather = sunny, weather = cloudy, and determining on the basis of observation the logical category which best fits the observation. Descriptive generalization is the grouping of a set of observations into a descriptive group based on some common characteristics. For example, based on the height measurements of a college basketball team, we might reach the conclusion that most basketball players are tall.

3. Machine Learning and Machine Discovery

a. Machine Learning

Machine learning is a sub-field of artificial intelligence dealing with the computer supported derivation of domain models which are based on knowledge representation paradigms of artificial intelligence. [4] Machine learning attempts to automate the learning process to discover new facts, concepts, and theories given a particular domain or area of interest. The machine learning technique is usually based on a representative human learning paradigm such as inductive inference.

b. Machine Discovery

Machine discovery is a sub-field of machine learning. It is a specialization of machine learning in that it is specifically concerned with the discovery of previously unknown information from the provided data or information which of value to the user. A common machine discovery learning paradigm is concept acquisition (learning by example). In this paradigm, given a set of facts F, and a predetermined set of classes S, the discovery method attempts to associate some subset of facts $f \supseteq F$ with a given class $s \supseteq S$. This is also referred to as learning by supervision, in that user supervision is required to identify the relevant classes of interest.

4. Knowledge Discovery in Databases

a. Basic Concepts of Knowledge Discovery in Databases

Knowledge discovery in databases (KDD) is the application of machine discovery methods to information which is stored in databases. Knowledge discovery is defined as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data."[1] KDD thus involves finding implicit patterns in the data which otherwise may go unnoticed. KDD discovery methods make use of the inductive inference process by associating sets of facts in the database with particular class outcomes; the discovered association is termed a pattern. Given the general nature of the process, KDD can be accomplished through a variety of discovery methods. Piatetsky-Shapiro, et al [1] cite four main characteristics displayed by knowledge discovery in databases.

1. High Level Language. Discovered Knowledge is represented in a high level language. It need not be directly used by humans, but its expression should be understandable by humans.

2. Accuracy. Discoveries accurately portray the contents of the database. The extent to which this portrayal is imperfect is expressed by measure of certainty. The patterns induced by the discovery methods will very seldom be exact due to values entered in error, missing information, and normal variation in the data.

3. Interesting Results. Discovered knowledge is interesting according to user-defined biases. In particular, being interesting implies that the patterns are novel and potentially useful. Knowledge is useful when it can help achieve a goal of the system or the user. The biases or preferences of the user may preclude some patterns with high certainty from being considered interesting results.

4. Efficiency. The discovery process is efficient. Running times for large sized databases are predictable and acceptable.

b. Database Issues

A database is a collection of related files. The most basic database model is based is on the aggregation of records associated with a particular transaction or event. Each record is referred to as a tuple. A data dictionary provides the definition of each field within a tuple and allowable values for an individual field. The database management system provides a means of storage, update, and retrieval of information for a particular event or group of events.

Database users typically have some domain knowledge learned through personal experience and consultation with the data dictionary. The user can utilize this learned knowledge to focus the search for patterns of interest. The use of this domain knowledge to focus the search for patterns of interest is controversial. [1] It is controversial because the use of the domain knowledge will focus the search in a particular direction and possibly reduce chances for the discovery of new and useful patterns. For instance, Structured Query Language (SQL) is an efficient means for data retrieval to support theories in the deductive inference process. It is not primarily designed to discover previously unknown information Unsupervised searches will improve chances of discovery of new patterns, albeit at a computational penalty.

Databases are typically designed to facilitate records keeping of information for an organization; application of KDD methods may come as an afterthought. As such, databases may contain irrelevant attributes and/or attributes entered in error. This is referred to as 'noise' in the data and is analogous to noise received in communications signals in that it may cause erroneous results. Missing values also may occur and affect outcomes of the KDD process.

Databases may only contain only sparse information with which patterns can be inferred. This sparse information may not be enough to provide a description sufficient to induce a reliable model.

B. KNOWLEDGE DISCOVERY METHODS

This section provides an overview of Genetic Algorithms and Genetic Programming that are basis of the knowledge discovery methods studied in the research.

1. Genetic Algorithms

a. History

Genetic algorithms are search algorithms which use the mechanics of natural selection and reproduction as their controlling mechanism. Genetic algorithms were developed by Holland (1975). They draw their logical basis from similarities to adaptive models in the biological world. Genetic algorithms are based on the darwinian principle of survival of the fittest in that only the most fit individuals in a species will be successful in finding food, water, and shelter to enable them to survive and reproduce. This analogy is used in grading a series of potential solutions to maximize or minimize an objective function corresponding to a solution to a problem of interest.

b. Theory of Genetic Algorithms

Goldberg [5] asserts that genetic algorithms (GA's) differ from other search techniques in four fundamental ways:

1. GA's work with a coding of the parameter set, not the parameters themselves.

2. GA's search from a number of points, not a single point.

3. GA's use payoff (objective function) information, not derivatives or other auxiliary information.

4. GA's use probabilistic transition rules, not deterministic rules.

A typical GA is represented as a string of binary digits. The objective function for the radius of a quarter arc of a circle $x^2 + y^2 = r^2$ over the interval $0 \le x \le 4$ could be represented by a four bit binary string such as [1011]. The bits of the string represent the measurement of the radius plus a constant one added to the value of the string. Thus, the string [1011] represents the value of a radius squared (r^2) with measurement of 12. Figure 2-1 is a graph of problem domain.

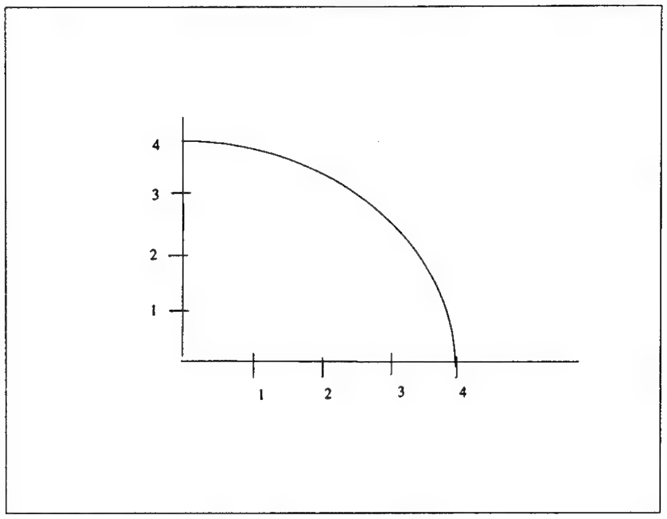

Figure 2-1.

The most highly fit string would represent the closest to optimization of the function in the problem domain. For this problem, the binary string with a value of sixteen [1111] (string value of 15 plus a constant 1) would represent the optimal value. Each string is graded according to a fitness function which measures the ability of the string to solve the problem at hand. A number of different strings, which is commonly referred to as the population, is used to conduct the search in a parallel manner. A fitness rating is developed for each string in the population, and the fitness of the population as a whole is computed. Selection of the next generation of strings occurs according to a probabilistic function based on the fitness of the strings. Highly fit strings have a better chance for selection and resulting survival than strings with comparatively lower fitness.

c. Reproduction, Crossover, and Mutation in Genetic Algorithms

Genetic algorithms typically use three operators which are Reproduction, Crossover, and Mutation.[5] Reproduction is simply selection of the most fit strings. A common method for implementation is the use of random selection in the form of a lottery with the more highly fit strings receiving more tickets or chances to be selected. In the previous example, assume a population of four strings. Table 2-1 shows each string with its fitness computed; the probability for selection for reproduction is based upon a particular string's fitness as a percentage of the total fitness of the population. The fitness of each string is simply interpreted as its decimal value, e.g., [0011] is equal to three, with no strings exceeding a value of sixteen in the problem domain.

String #	String	Value of Fitness	% of Total
S_1	[1011]	11	39%
S_2	[0011]	3	11%
S_3	[1000]	8	29%
S_4	[0110]	6	21%
Total		28	100%

Table 2-1

A possible result of the reproduction selection process is displayed in Table 2-2 , where string S_1 has been selected twice and strings S_3 and S_4 have been selected once.

String #	String	Value of Fitness
S_1	[1011]	11
S_1	[1011]	11
S_3	[1000]	8
S_4	[0110]	6
Total		36

Table 2-2

Reproduction alone improves the number of highly fit strings in a population; it may not provide for optimization of the desired function in that it does not provide any improvement to existing strings in the population.

Crossover is the genetic mating of two strings in an attempt to create a more highly fit string from the genetic material of their ancestors. Crossover occurs by the random selection of a crossover position *k* where *k* is a position between 1 and *l* - 1; *l* is the length of the string and 1 is the first position of the string. The subsequent exchange of bits between two strings takes place at positions *k* +1 and l inclusively.

For population two, crossover was randomly selected to occur at position k=1. Strings S_1 and S_4 were randomly chosen for mating with the results displayed as Table 2-3. Crossover between string S_1 [1011] and S_4 [0110] at position k=1 results in the swapping of bits between S_1 and S_4 at the second leftmost position in the strings; the new value of S_1 is now [1110] with a value of [0011] for S_4.

String #	String	Value of Fitness
S_1	[1110]	15
S_1	[1110]	15
S_3	[1000]	8
S_4	[0011]	3
Total		41

Table 2-3

Reproduction and crossover have resulted in an increase in the fitness of the population and the result of a more highly fit string than previously encountered in the prior population. Genetic Algorithms improve searches by (1) reproducing high quality notions according to their performance and (2) crossing these notions with many other high performance notions from other strings [5]. A specified number of generations of evolution for the GA process are selected by the user.

Mutation is a random operation involving the alteration of a single bit or bits in the binary string. Mutation provides a means of change for a string or strings when all the strings in the population are converging to a local (but not global) maximum.

Both the string position and population elements selected for mutation are random. The implementation of the mutation operation may or may not improve the performance of the string. Empirical genetic algorithm studies suggest that one mutation per thousand bit (position) transfers may produce good results. Mutation rates are similarly small (or smaller) in natural populations leading many researchers to conclude that mutation may be considered as a secondary mechanism of genetic algorithm adaption. [5]

2. Genetic Programming

Genetic Programming (GP) is a logical extension of Genetic Algorithms developed by Koza. [6] In contrast to the fixed length strings used by Genetic Algorithms, GP uses actual program statements, in the form of parse trees, to evolve a program which would provide a working solution to some problem of interest.

a. Genetic Program Constructions

Koza uses a type of expression in the implementation of GP known as a symbolic expression or S-expression. The S-expression is an integral part of the LISP programming language. A LISP expression, such as, "$a * (b+c)$" is equivalent to a parse tree for a section of a program. Such as a statement would appear in parse tree format as depicted in Figure 2-2.

The unique capability of the LISP S-expression to serve as both data and as an executable program statement was one of the key reasons LISP was chosen for implementation of GP. [6] However, it is also possible to write genetic programs with standard procedural languages such as C and Ada.

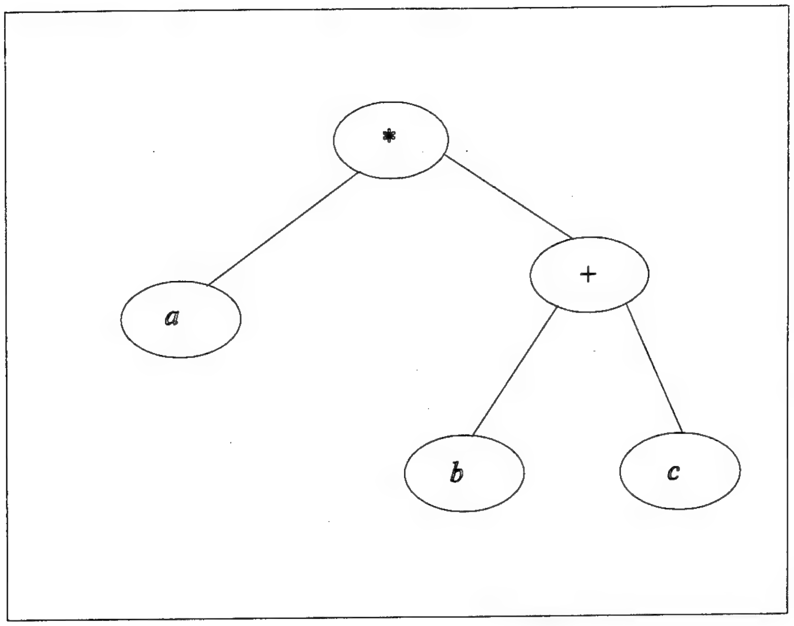

Figure 2-2.

The LISP S-expressions used in GP fall into two major categories which are known as the function and terminal set. The function set consists of domain specific functions which the user selects as appropriate for solving the problem. Examples would be boolean, arithmetic, and other mathematical operators such as exponential functions. The terminal set consists of arguments to the function set during the evolution of the genetic program. The terminal set usually consists of relevant variables from the problem domain.

An important feature which genetic programs must incorporate is the closure property. This property prevents the evolved genetic program from taking on illegal arguments which would result in the execution of a program statement ending in an error. A classic example of this is a program which attempts division by zero. The closure property requires definition in the function set of a special division operator to prevent this situation from occurring.

b. Operation of a Genetic Program

Genetic programs use the genetic algorithm concepts of reproduction, crossover, and mutation. The initial population consists of a series of randomly generated programs. As in genetic algorithms, evolution of a number of populations of programs occurs. The crossover operation involves the swapping of S-expressions between programs in order to evolve a more fit single program.

c. Application of Genetic Programming to KDD

Genetic programming methods can be applied to a wide variety of machine discovery problems, including knowledge discovery in databases. The fitness function used by the GP program to implement KDD would be one designed to favor evolution of programs which can locate patterns in the database. The data mining software used in this study which is known as the Naval Postgraduate School Genetic Program is an example of such a system.

C. REPRESENTATION OF DISCOVERED KNOWLEDGE

Discovered knowledge in the KDD process may be represented in the form of rules. Rules represent knowledge in a form humans can readily understand and use. A rule language serves two general purposes. The first is to express discovered knowledge found by the discovery method. The second purpose is to serve as a prediction language. [7] Extensions of rules with basic probability concepts allow prediction of outcomes when previously unseen test data is encountered. The predictions are based on previous outcomes which served as the basis for rule generation.

1. Representation by Rules

a. Production Rules

Rules induced from databases have been traditionally placed in the form of production rules which are understandable and usable by humans. A production rule is interpreted as a condition-action pair with the context If condition A, Then action B. Given a set of multi-valued attributes $\{a_1, a_2, ..., a_n\}$ and a series of distinct classes c_n, an elementary description takes the form If $a_1 \wedge a_2 \wedge ... \wedge a_n$, then c_1, where a_1, a_2 ... take on some attribute value condition and c_1 is a class outcome.

Rules can also take on disjunctive conditions such as If $a_1 \wedge a_2 \vee (a_3 \wedge a_4)$, then C_1. Rules are typically expressed with some degree of certainty expressed by a probability p(x). Rules induced from real world databases are rarely exact due to noise from erroneous values and normal variation in the data.

From Carbonell [3], the four basic operations whereby production rules may be acquired and modified are :

1. Creation. A new rule is created by the system or acquired from an external entity.

2. Generalization. Conditions are dropped or made less restrictive, so that the rule applies in a larger number of instances.

3. Specialization. Additional conditions are added to the condition set, or existing conditions are made less restrictive so that the rule applies to a smaller number of specific conditions.

4. Composition. Two or more rules that were applied together in sequence are made into a single larger rule, thus forming a "complied" process and eliminating any redundant conditions or actions.

b. Decision Trees

Rules induced by the discovery method can also be portrayed in the form of Decision trees. A tree-like structure is produced with attributes represented as nodes from which further branching can occur. Individual classes are represented as leaves of the tree and may be further described using class probabilities or actual class counts.

2. Rule Utility

Given a set of patterns (rules) induced from a database, some rules will be more useful than others to the user. This is the notion of rule utility. The utility, which is generally defined as usefulness of the rule to the user, is a subjective concept and will vary among users for a particular rule. The utility of the rule applies to the application of the rule in the context of the set of possible actions. A rule with high utility will enable an action to be taken which results in a gain or the avoidance of a loss. In the section, we discuss rule interestingness, a measure of rule utility.

a. Rule Interestingness

Given a set of induced rules, some rules will be superior to other rules in terms of simplicity, coverage, and accuracy of the rules. Rule Interest (RI) measures provide a means for grading of rules in accordance with some predefined criteria. The rule interest measure is typically a statistics or information theory concept. RI measures are useful for comparing the output of different rule induction algorithms, and they can be applied also to the ranking of rules previously identified as rules having some utility to the user. Rules with low coverage and certainty will not be of much use to the user and therefore tend to be of low interest.

b. General Measures of Rule Interestingness

The most basic rule interest measure used is referred to as the Certainty measure. The certainty measure is simply the number of correct classifications of the rule divided by the number of examples in the database. Piatetsky-Shapiro [8] suggests that all rule interest measures should satisfy several basic principles. Rules satisfying these proposed principles should assign high values to strong rules and lower values to weak rules. His proposed measure of rule interest is described as follows:

Let N indicate the number of tuples in a database of interest. Let |A| be the number of tuples satisfying condition A and let |B| be the number of tuples satisfying condition B, and let |A&B| be the number of tuples with the condition A→ B. The proposed rule interest measures are:

1. $|A\&B| = \frac{|A|\ |B|}{N}$, if A and B are statistically independent, the rule is not of interest.

2. RI ↗ (monotonically increases with) |A&B| when other parameters remain the same.

3. RI↘ (monotonically decreases with) |A&B| when other parameters remain the same.

Piatetsky -Shapiro suggests the simplest function which meets these three criteria is

$$|A\&B| - \frac{|A|\ |B|}{N} \qquad \textbf{(2-1)}$$

This is the difference between the number of tuples containing |A&B| and the number of tuples of |A| and |B| which would be expected if A were independent of B.

Intuitively, RI depends on the coverage of the rule, i.e., what proportion of the database does the rule classify and the certainty of the relationship, i.e., what is the correlation of a particular set of attributes A with the predicted class B.

c. An Information Theoretic Approach to Rule Interestingness

Goodman and Smyth [9] suggest an information theoretic approach to evaluating rule interest. Given a rule, with some set of attributes X and a set of object classes Y, in the form if (X=x), then (Y=y) with some probability p(a), then some information is provided by Y=y about X. This information is used in part for the evaluation of rule interest.

Goodman and Smyth propose a rule interest measure known as the J-measure. [9] J-measure incorporates the simplicity of the hypothesis and the goodness of fit of the class to the attributes to come up with a measure of rule interest. It satisfies the concept of average mutual induction between discrete random variables as originally defined by Shannon. The J measure formula is defined as follows:

$$j(X;Y=y) = p(y)\ p(x|y)\ \log \frac{p(x|y)}{p(x)} + (1 - p(x|y))\ \log \frac{(1-p(x|y))}{(1-p(x)\ )} \qquad \textbf{(2-2)}$$

The J measure can be broken into two fundamental parts. The first part p(y) expresses the coverage of the class y in the database. The remainder of the J-measure is known in information theory as the cross-entropy of X on the condition that Y=y. Cross-entropy is a distance measurement between the a posteriori belief about X and the a priori belief concerning given a particular outcome of Y=y. The log base of the cross entropy gives less weight to rare events (circumstances where p(x) is low) as compared with Piatetsky-Shapiro's Rule Interest measure. J measure thus has a preference for events which occur frequently in the database and also have a strong correlation between X and Y. J measure can be used to rank rules by information content which appear to be redundant with respect to the same outcomes with the lower ranking rules being eliminated.

Goodman and Smyth explicitly comment that "To a large extent, ... information based and correlation based measures, in practice, often rank rules in a similar order." [12] This is not a surprising result since the concept of mutual self-information is based on the conditional probabilities between series of events in a system albeit used in an informational (log based) manner. [13] Specifically, the mutual information I(E_j , F_k) between the two events E_j and F_k is defined by:

$$I(E_j, F_k) = \log P \frac{(E_j \cap F_j)}{P(E_j)\, P(F_k)} = \log \frac{p_{jk}}{p_j q_k} \qquad \textbf{(2-3)}$$

The mutual information in a system would involve a summation of the mutual information for all values of E_j and F_k.

3. Strategies for Refinement of Rules

The discovery method used to extract patterns from a database may discover a large number of patterns. Many of these patterns may be redundant or so specialized that they are not immediately useful. The user of the knowledge discovery method would like to choose the most general and useful patterns for his own use. A coherent strategy must then be developed for selection of the best patterns with regards to certainty, coverage, and novelty of the pattern.

a. *Production Rule Refinement*

Major and Magano [11] suggest a rule refinement strategy to eliminate rules with low interestingness. Interestingness is defined according to four criteria of performance, simplicity, novelty, and significance. Rules which are not interesting according to these criteria are dropped from the set of rules. The strategy is implemented as follows:

> 1. Performance. A performance frontier is defined consisting of a Cartesian plane with axes of < coverage, certainty factor >. If Rule1 is at <G, C>, and Rule2 at <F, B> with G > F and C > B, then Rule1 is said to dominate Rule2.
>
> 2. Simplicity. The concept of a rule lattice is introduced to provide an order among different rules. Simpler (more general) rules subsume more specialized rules if they both cover the same concept. A rule with no further generalization is said to be a Most General Rule (MGR). Rules which provide specializations of the MGR are said to be in the family of the MGR.
>
> 3. Novelty. In a situation where Rule1 and Rule2 have overlapping concepts, one of the rules will most likely be preferred to the other on the basis of scientific rationale, uniqueness, or performance. "A rule that adds little insight or performance to an existing set of rules has no novelty with respect to that set." [11]
>
> 4. Significance. For a rule to be considered interesting, it must vary significantly from other rules in the rule set. A significance measure was devised to take into account the complexity of the rule and the statistical significance based on an approximation of the Chi-Square test. The complexity measure *J* for a rule for a more

general concept *Q* and a rule *R* for a specialization of *Q* is:

$$J = \left(\frac{I}{G}\right)\left(\frac{V!}{(T!\ (V-T)!)}\right) \qquad \textbf{(2-4)}$$

where I is the number of instances of concept Q, G is the number of instances covered by concept R, V is the number of variables available to formulate specializations of concept Q, and T is the number of conjuncts present in R. J is then the specializations of the more general concept Q examined by the induction mechanism.

The significance is measured by:

$$S(R|Q) = -\log_{10}(A\ J) \qquad \textbf{(2-6)}$$

where A is a numerical approximation to the one tailed significance level of the Chi-Square test. An arbitrary cut-off of 2.0 was established for the domain researched by the authors although this could vary based on the needs of the user. The significance measures favor rules with statistically significant differences and simplicity in the structure of the rule.

Major and Magano define potentially interesting rules are those that satisfy the performance criterion or are closely related to rules that do. A rule R is potentially interesting if :

1. R is on the performance frontier or

2. Q is on the performance frontier and R is in the cone of Q or

3. Q is a most general rule and there are at least three frontier rules in Q's family, and R is also in Q's family

Technically interesting (TI) rules are selected among potentially interesting rules according to simplicity and statistical significance criteria. A rule R is technically interesting if R is potentially interesting, and for all Q that Q is TI and R specializes Q and S(R|Q) > 2.

Rules which are not Genuinely Interesting are then removed. This involves examination of the TI rules which to pick the rules which are most applicable to the concept of interest. Redundant rules which do not provide an increase in performance are removed from the rule set. A domain expert is used to pick the rules which are the most useful and relevant in a scientific sense. The goal is to reduce the induced number of rules to small number of rules with high coverage, certainty, and applicability to the concept under consideration.

III. TESTING OF INDUCED RULES

A. RELIABILITY OF INDUCED RULES

Machine Discovery methods such as Genetic Programing can be used by managers to find patterns in data bases in the form of rules. The rules are graded according to rule interest measures as previously discussed. Before the rules are used for actual decision-making purposes, testing of the rules must occur to ensure that the induced rules are based on valid statistical relationships. The use of untested rules would represent a risk by acting on information which, in fact, may be the result of random variation in the data used to induce the rule being evaluated. Hence, statistical procedures are a necessity for assessing the reliability of rules.

1. Statistical Testing

Mathematical Statistics is the field of science which deals with the mathematical treatment of random occurrences. Mathematical statistics comprises probability theory, statistics, and their applications. A general application of mathematical statistics is to assess the accuracy of models which have been inferred from an inductive process. Probability theory allows for the assessing of the accuracy of a model based on known information. The testing of models developed inductively by deductive reasoning in the form of statistical testing is necessary to develop accurate models. This forms the basis of the scientific method; a model developed inductively is tested with what information is known in regards to probability of the model. The process is depicted in Figure 3-1.

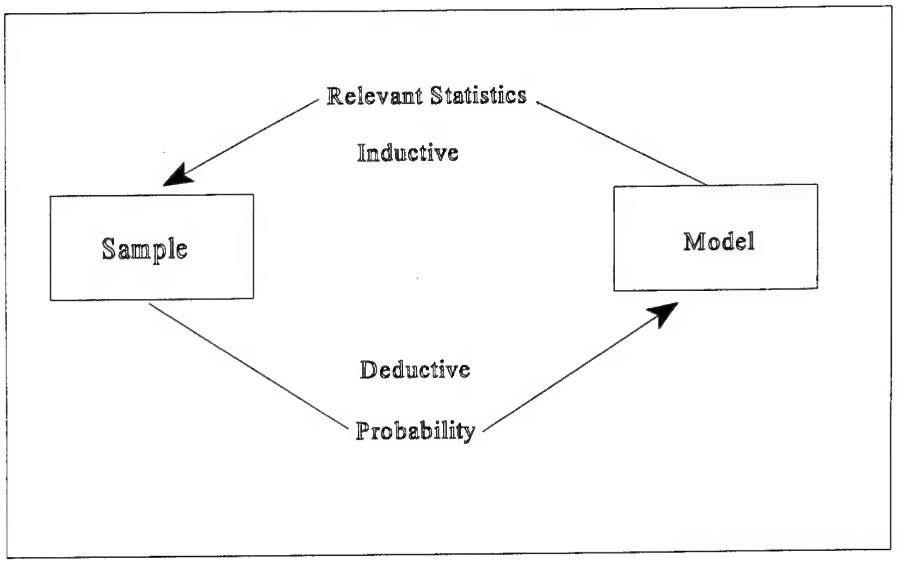

Figure 3-1. After Ref. [12]

2. Hypothesis Testing

Hypothesis testing is used to see if two populations agree on some common parameter. Normally, the hypothesis that two populations agree is termed the null hypothesis and is denoted by H_0. The hypothesis that the populations do not agree is termed the alternative hypothesis and is denoted by H_A. The test has two possible outcomes; H_0 is accepted, or H_0 is rejected. Since hypothesis testing is normally conducted with samples from two different populations, there will likely be differences in the parameter being tested between the two populations. Rejection of the null hypothesis will occur when results are received that are of low probability under the null hypothesis.

The problem of what constitutes a low level of probability is covered by the notion of significance level. Sachs [12] addresses the concept of significance level.

If a test with a level of significance of, for example, 5% (significance level $\alpha = 0.05$) leads to the deduction of a difference, the null hypothesis is rejected and the alternative hypothesis -the populations differ- is accepted. The difference is said to be important or statistically significant at the 5% level, i.e., a valid null hypothesis is rejected in 5% of all cases of differences as large as those observed in the given sample, and such differences are so rarely produced by random processes alone that: a. we will not be convinced that random processes alone give rise to the data or, formulated differently, b. it is assumed that the difference in question is not based solely on a random process but rather on a difference between populations.

3. Problems with Testing of Rules Induced by Genetic Programming

Genetic Programming heuristically searches for patterns corresponding to high levels of the specified fitness function. The program searches in parallel from a number of starting points and each member of the population evolves through crossover with other members of the population through successive generations. The output of the genetic programming process is a number of multiple independent models in the form of rules corresponding to patterns in the data base. Jensen [13] notes that problems occur when testing multiple models. Testing multiple models increases the probability of finding an apparently accurate model by chance alone.

a. Labelling Spaces

Jensen illustrated the problem with testing multiple models through the use of labelling spaces. A labelling is a set of class observations with each observation corresponding to a label. The labelling space *L* is an *N* dimensional space where *N* is the number of observations. Any one labelling represents a point in the space of all possible labellings. The scoring statistic, which is the fitness function in the case of rule induced from genetic programming, represents the distance *d* between the actual and predicted labellings. Thus, a labelling for a rule with a high fitness value would have a smaller distance *d* than a labelling for a rule with a lower fitness value. Significance testing can be viewed as determining the probability *p* that an arbitrarily chosen labelling fall within distance *d* of the predicted labelling. All such labellings fall within the shaded area depicted in Figure 3-2. The probability *p* is equal to the ratio of the number of labellings within the shaded area to the number of labellings within the entire labelling space *L*.

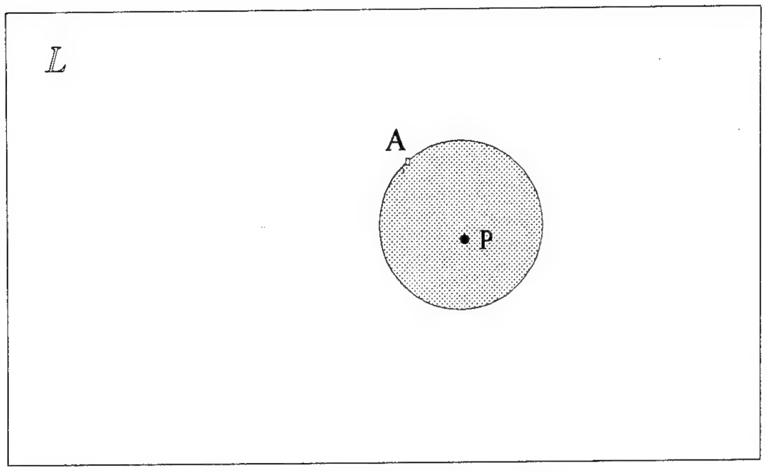

Figure 3-2 After Ref. [13]

b. Testing of Multiple Models

The testing of multiple models is problematic in that it increases the chances of finding an apparently accurate model by chance alone. This concept can also be illustrated by labelling spaces as depicted in Figure 3-3. The points *P1* ... *P5* represent individual models in the labelling space.

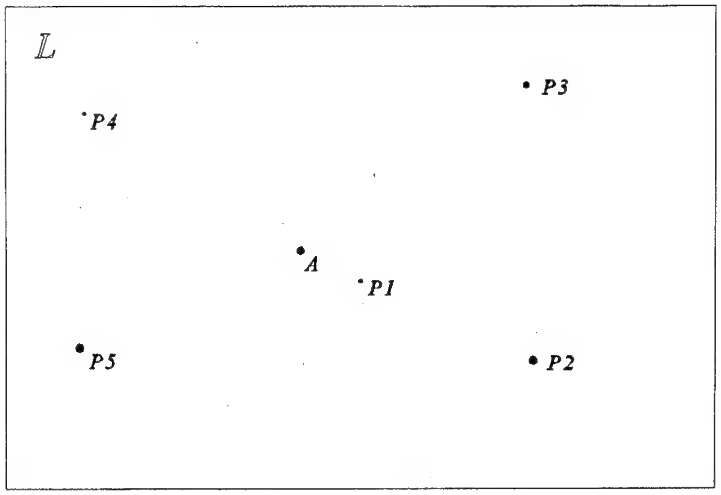

Figure 3-3 After Ref. [13]

When multiple models are tested, significance testing can be viewed as determining the probability that an arbitrarily chosen label falls within distance *d* of any of the predicted labellings. The distance *d* is the actual distance between the actual Labelling *A* and the best of the predicted labellings (*P1*). [13] The distance *d* is then used as the radius for the shaded circles in Figure 3-4.

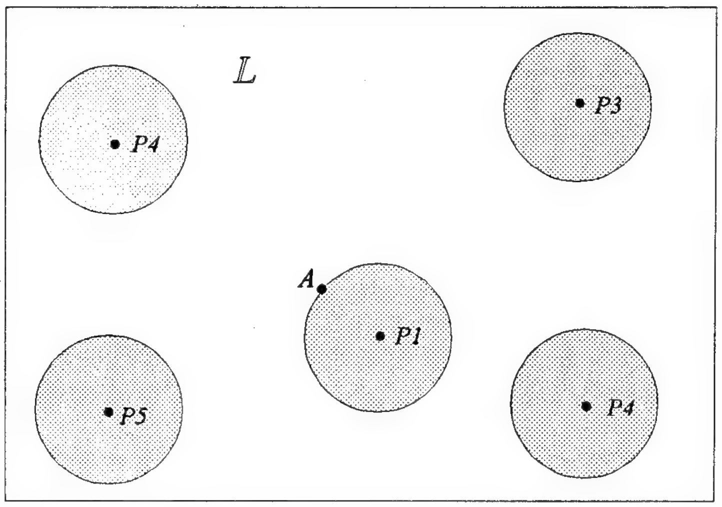

Figure 3-4 After Ref [13]

More area is clearly in shade in Figure 3-4. This increased area represents the probability that a particular model will be found to be statistically significant. Jensen comments:

> Testing multiple models puts more labellings within reach of at least one of the models. This increases the probability that an apparently accurate test will be found by chance alone. [13]

Conventional parametric statistical tests are designed to test one model at a time and assume that it is the only model tested. For this reason, use of a conventional statistical test could lead to misleading results when testing rules obtained from genetic programing.

c. *Testing of Correlated Models*

Jensen also notes that testing of results from induction systems also introduce an additional difficulty due to the testing of multiple models with highly correlated predictions. High correlations between models may result from the models being closely related in form or from the models being induced from the same data set. Correlation between models decreases the probability of finding an apparently accurate model from chance alone. [14] [15] Jensen also explains this effect in terms of labelling spaces. Highly correlated predictions are closely grouped in the vicinity of the actual value *A*. The most accurate model has the labelling *P1*. This is depicted in Figure 3-5. Significance testing is used to determine the probability that an arbitrarily selected labelling would fall within distance *d* of one of the models. The correlated labellings reduce the amount of labellings within reach of each individual model, thus lessening the possibility that a spurious labelling may be included in a particular model. This is depicted in Figure 3-6.

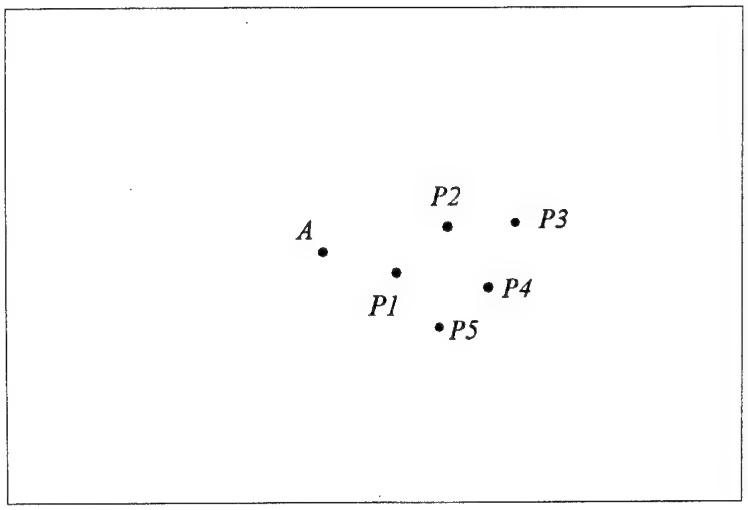

Figure 3-5 After Ref. [13]

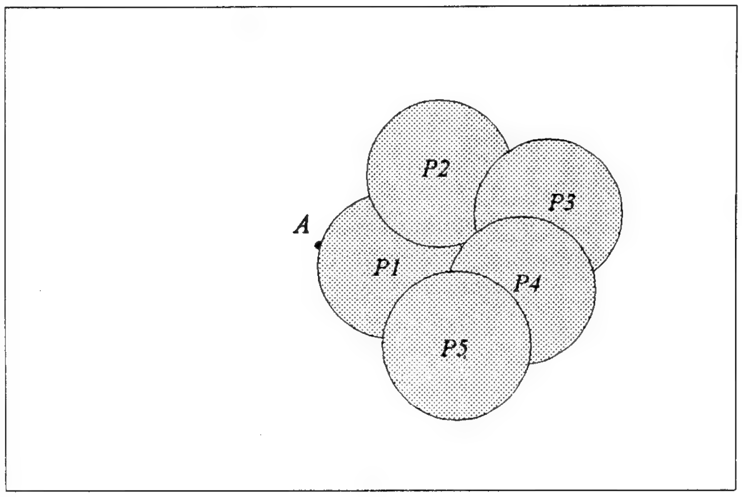

Figure 3-6 After Ref. [13]

B. RANDOMIZATION TESTING

Randomization Testing is a non-parametric statistical test which can be employed for hypothesis testing of differences between two populations. Randomization testing is also referred to in the literature as permutation tests. [16] Randomization testing is attractive for testing rules induced from genetic programming because it makes no mathematical assumptions concerning the number of models tested or the correlation of the predictions of the models. As per Jensen [13], "Randomization tests have several advantages over other approaches. They automatically account for the number of observations, the number of models tested, and the correlation of the models." Edgington [17] extensively covers the applications of randomization testing to a variety of problems.

1. Randomization Testing Model

The randomization test is a variant of Fisher's Exact Test (1935). It is a rank order statistic method based on a comparison of the actual experimental results with a randomly derived set of results. Edgington [17] formally defines the randomization test as:

A statistical test for which the significance is determined by permuting the data repeatedly to compute *t*, *F*, or some other test statistic is called a randomization test.

The idea and methodology behind the randomization test is relatively simple. First, a test statistic relevant to the problem is selected. The test statistic is then computed for the model based on the actual data set. The data is permuted in a random fashion and the test statistic computed for each permutation of the data. The test statistic for the actual (non-permuted) data is combined with the test statistics from the random permutations. The resulting distribution of test statistics is then placed in rank order. To reject H_0, and conclude that the sample from the actual results did not come from the sample of randomly permuted results, the performance of the actual results must outrank the majority of the randomly permuted results. Test statistics from the random permutations represent the best which can be achieved by chance alone. If the test statistic from the actual data set does not outperform the randomly derived test statistics, it can be construed as evidence to support the null hypothesis that there no differences between the results of the model being tested and the results which could be expected from a random process.

a. Randomization Test Procedures

As the randomization test is designed to test the differences between two populations, the hypothesis is stated in a standard manner which is similar to the notation used in conventional two-tailed statistics tests. Edgington [17] discusses test design implementation of randomization testing. The null hypothesis is that the measurement or set of measurements associated with each experimental unit is independent of the assignment of units to treatments. In the case of rules derived from genetic programming methods, this means that the performance of a rule on some test statistic is not significantly better than could be achieved by random. The steps to be taken to ensure validity of the test are as follows:

1. Specify the test statistic before the experiment, and ensure that the test statistic is defined in such a way that it can be computed for a data permutation without considering whether the data permutation represents the obtained results.

2. List every equally probable assignment of experimental units to treatments. (This step presupposes a random assignment to treatments)

3. Within each of the equally probable assignments, substitute for each experimental unit the measurement (or set of measurements) for the unit to provide a distribution of data permutations.

4. For each data permutation compute the test statistic specified in step 1.

5. Determine the population of the data permutations with as large a test value as the obtained test statistic value, and use that proportion as the P-value.

b. Achieved Significance Level

The Achieved Significance Level (ASL) of the test procedure is a p value based on the proportion of data permutations with as large a test value as the actual obtained test statistic value. Efron [16] states the formal definition of the ASL as

$$ASL_{perm} = \frac{\#\{\, \hat{\Theta}^{*} \geq \hat{\Theta} \}}{B} \tag{3-1}$$

where $\hat{\Theta}^{*}$ is the test statistic from the random permutations and $\hat{\Theta}$ represents the test statistic from the actual data set. B is the number of instances of test statistics in the distribution. To reject H_0, and conclude that the test statistic derived from the actual data set did not occur as a result of a random process, $\hat{\Theta}^{*}$ should outrank $\hat{\Theta}$ 5 times for a B value of one hundred. This would correspond to a p level of .05 for the significance of rejecting the null hypothesis.

c. Random Data Permutation

Two methods exist for obtaining the random permutations of data used to conduct a randomization test. They are referred to as systemic data permutation and random data permutation. A systemic data permutation includes all permutations of the data in the randomized data set used to conduct the test. A random data permutation involves the sampling of permutations to obtain a randomized data set.

A problem which could occur with random data permutation method is that the sampled permutations are not representative of a uniformly random distribution. This would likely be the result of Monte Carlo error in the sampling procedure and be more prevalent with a small number of permutation replications. Efron [16] addresses the problem of the number of permutation replications to be used to minimize Monte Carlo error in replication sampling. Computations were performed to find the number of permutation replications required to make the coefficient of variation of the achieved significance level $_{cv}(ASL) \leq .10$. Efron's results are summarized in table 3-1.

ASL:	.5	.25	.1	.05	.025
B:	100	299	900	1901	3894

Table 3-1 From Ref [16]

The next chapter discusses the use of randomization testing for assessing the statistical significance of rules induced from genetic programming.

IV. DATA AND METHODOLOGY

A. NAVAL POSTGRADUATE SCHOOL GENETIC PROGRAM

The Naval Postgraduate School Genetic Program (NPSGP) is an adaption of the Simple Genetic Program in C (SGPC) written by Tackett and Carmi. NPSGP modifies SGPC by providing functionality for data mining. Only a brief description of the preparations for use of NPSGP is provided here, a full User's Manual is also available. [18]

1. Preparations to use NPSGP

a. Data Requirements

The data set intended for data mining with NPSGP must be in a tab-delimited ASCII file. Additionally, NPSGP works best if the following conditions are met:

1. The target attribute (classification) is represented by non-continuous (discrete) data.

2. The non-target attributes in the data set are represented by a mix of continuous and discrete attributes.

3. All continuous attributes are linearly scaled to the same range.

b. Fitness Functions

The fitness function in NPSGP must be translated in C code to correspond to a particular rule interest measure as defined in Chapter II. For each rule evaluated, NPSGP partitions each example in the data set into one of four possible conditions. The conditions are displayed in Table 4-1.

Classification (RHS) is :		
	True	False
Description (LHS) is True	a	b
Description (LHS) is False	c	d

Table 4-1

The rule interest measure is then coded to match the components of the table. For example, the fitness function for the certainty rule interest measure would be simply
(a / (a + b)).

c. *Description of NPSGP Rules*

A typical production rule induced from a data set used in the study would appear as:

```
Tree # 45
( IF
   (IN-CTGRY
        GILL_SIZE
             N     )
   (IN-CTGRY
        CLASSIFY
             P  )
                    )
Number of Records matched by LHS: 2512
```

The rule was developed from a data set of mushrooms used in the study. It is read as "IF (the example mushroom) is in category of the attribute GILL_SIZE = N, THEN the classification of the mushroom is P (poisonous). Information is also given on the number of examples in the data set matching the on-target attribute description of the rule and the number of times the rule provides incorrect classifications.

B. DATA SELECTION

Two different data sets were used for generation of rules corresponding to patterns in the data set for testing purposes. Both data sets are commonly used for evaluation of machine learning methods by researchers. The data sets are available through anonymous ftp from the University of California - Irvine (ics.uci.edu: pub/machine-learning-databases).

1. Description of the Mushroom Data Set

The mushroom data set was donated by Schlimmer and contains 8124 examples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota families. Each tuple contains 22 nominally valued attributes and a classification for the example. Each example is classified as definitely edible, definitely poisonous, or of unknown edibility. The latter class was combined with the poisonous one. A full description of the data set and domain information is provided in Appendix A. Ref. [18] clearly states that there is no simple rule for determining the edibility of a mushroom; no rule such as "leaflets three, let it be" for poisonous Oak and Ivy.

a. Modifications to the Mushroom Data Set

Modifications were made to the mushroom data set to improve the performance of the NPSGP data mining software used in the study. Two nominally valued attributes (Gill Spacing and Ring Number) were reformatted with continuous values over the range of 0 -100 to provide a mixture of discrete and continuous attributes to enhance performance. Additionally, the data set was modified to test the ability of the software to find a rare event. To facilitate this, a high quality rule in the data set was identified, (If Odor = none, Then mushroom is edible), and the 852 examples supporting this rule were changed from a classification of "e" to a classification of "z". The actual rules tested for significance from this data set were from a related study at the Naval Postgraduate School. [19]

2. Description of the Zoo Data Set

The zoo data set was donated by Forsyth and contains 101 examples of zoo animals which are divided into seven distinct classes. The classes are standard taxonomic categories such as mammals, reptiles, birds, etc. All non-target attributes are boolean with the exception of one continuously valued attribute. The details of the attribute names and values are provided in Appendix B. The actual rules tested for significance from this data set were from previous research at the Naval Postgraduate School. [20]

C. RANDOMIZATION TESTING DESIGN

The design of the randomization testing to test the significance of the rules induced from NPSGP involved the following steps:

1. Statement of the Hypothesis.

2. Computation of the test statistic of the NPSGP rules on the actual data set. For the purposes of the study, the test statistic used was the confidence of the rule.

3. Randomization of the class labels in the data set.

4. Compute the test statistic of the NPSGP rules on the randomized data set.

4. Repeat steps 2 and 3 for the required number of permutation replications.

5. Compute the achieved significance level for each rule being tested.

6. Evaluate the stated hypothesis.

1. Statement of Hypothesis

The null hypothesis is that rules from induced by NPSGP are not more accurate than rules induced by random processes, or alternatively stated, there is no difference between the populations. The alternative hypothesis is that there exists a significant difference in the accuracy of rule induced by NPSGP as compared to the performance of rules induced by random processes. The hypothesis is formally stated as:

H_0: $F_1 = F_2$

H_A: $F_1 \neq F_2$

where F_1 represents a NPSGP induced rule and F_2 represents a rule that is formed as a result of a random process.

2. Randomization of the Data Set

The randomization test process requires the random assignment of class labels to the examples in the data set for each permutation replication. In the study, the classification labels were randomly assigned in the same proportions as they originally occurred in the data set, e.g., for the 3916 occurrences of the poisonous classification in the mushroom data set were randomly assigned to 3916 examples in the data set. This method involves sampling the total number of permutations of class labels. For the mushroom data set of 8124 examples, there are 23,372 permutations without replication, i.e., each example in the data set is assigned all three of the possible class labels with one the labels being the correct one. The sampling was done to preserve the continuity of results occurring under the randomization test If by chance the randomization process happened to correctly assign the

3916 poisonous class labels to the corresponding correct examples in the data set, then an equivalent test statistic should be computed for the actual and random results. Randomization of class labels was accomplished by a program written in C++ for the two specific data sets used in the study.

3. Generation of Permutation Replications

In a permutation replication, each NPSGP rule tested was scored on its performance is classifying the relevant data set with randomly assigned labels as previously described. The permutation replications were generated by a procedure which was executed in a Unix C shell script. The procedure is represented in pseudocode:

```
For j ... # number of replications, Loop

Randomize class labels in the data set

Compute confidence statistic for each rule tested

Record results for permutation replication on randomized data set
End Loop
```

A utility program was developed in C++ to grade the confidence of each NPSGP rule on the permutation replications. As per Efron [16], 2000 permutations replications were selected for each rule to minimize the coefficient of variance of the achieved significance level to less than or equal to 0.10.

4. Compute Achieved Significance Level

The computed test statistics for the NPSGP rules on the actual data set and on the permutation replications are arranged in rank order. The achieved significance level is computed by summing the number of times the test statistic score from the permutation replication exceeds the test statistic score achieved on the actual data set divided by the total number of test statistics.

V. ANALYSIS OF RESULTS

This chapter presents the results produced by the methodology described in the previous chapter. Three different sets of rules meeting the criteria of being technically interesting as defined in Chapter II were tested for significance using the randomization test methodology described in the previous chapter. Two of the sets of rules tested were from the mushroom data set and the other set of rules from the zoo data set previously described. A sample result of the testing method is shown by the example in Table 5-1. The significance level of the test in Table 5-1 is .05, but given the results of the test, the same conclusion with respect to the null hypothesis would have been reached for a significance level of .01.

A. SUMMARY OF RESULTS

1. Testing on Rules from the Mushroom Data Set

The first set of rules tested were from the mushroom data set after the NPSGP program was allowed to run for 30 generations. The twelve rules in the set were induced using the confidence fitness function. A summary of the rules' performance on the actual data set is presented in Table 5-2. The exact syntax of the rules and the C++ utility program used to test the rules is included in Appendix C. The scoring statistic for the 2000 randomization permutation replications is the confidence of the rule on each permutation replication. The scoring summary of the results of the 2000 permutation replications on the twelve rules is presented in Table 5-3.

The accuracy of the rules on the randomized data set consistently ranged between 0.40 and 0.50. None of the scores of the permutation replications on the confidence test statistic exceeded any of the results obtained on the actual data set. Therefore, the null hypothesis that the rules induced under genetic programming do not differ from those randomly induced is rejected. The consistency in the results of the permutation replications is mostly likely due to large number of examples (8124) and the distribution of the small number of classes (3) in the data set.

	Sample 1	Sample 2
Rule	If Bruises = True, Then Edible	If Odor = Foul or Gill Spacing >= 5.93 and Gill Spacing <= 62.05, Then Edible
Actual # of Tuples in Data Set	8124	8124
Actual # Left Hand Side (Attributes) matched in Data Set	3376	6812
Actual # Right Hand Side (Classification) matched in Data Set	2264	2430
Actual # of Misclassified Records.	1112	4382
Confidence of the Rule (%)	67.0 %	35.6 %
Actual # of Randomization Trials >= Confidence of the Rule	0	2000
Actual # of Randomization Trials < Confidence of the Rule	2000	0
Hypothesis Conclusion at Significance Level $p = .05$	Reject null hypothesis	Fail to reject null hypothesis

Table 5-1

Rule Number	LHS matched	# misclassified	Confidence (%)
1	2372	144	93.9
2	556	44	92.0
3	2512	288	88.5
4	3188	808	74.6
5	828	228	72.4
6	2480	720	70.9
7	2304	144	93.7
8	512	64	87.5
9	1872	576	69.2
10	1024	120	88.2
11	3376	1112	67.0
12	1640	288	82.4

Table 5-2

Rule Number	Count >= .50	Count < .50 and >= .40	Count < .40
1	95	1905	0
2	0	1830	170
3	30	1970	0
4	21	1979	0
5	371	1629	0
6	73	1927	0
7	110	1890	0
8	509	1491	0
9	130	1870	0
10	190	1810	0
11	0	1979	21
12	75	1925	0

Table 5-3

The second set of rules tested were also from the mushroom data set. However, this set of eight rules were selected from the first generation of rules produced from the NPSGP program and are generally of lower quality on the confidence fitness function. The performance of this set of rules on the actual data set is presented in Table 5-4. The exact syntax of the rules and the C++ program used to test the rules is included in Appendix D.

Rule Number	# LHS matched	# misclassified	Confidence (%)
1	1072	400	62.7
2	1872	576	69.2
3	1202	410	65.9
4	1048	291	72.2
5	5612	2505	55.4
6	7004	3344	52.3
7	4936	2232	54.8
8	6476	3369	48.0

Table 5-4

Table 5-5 lists the results obtained from 2000 permutation replications.

Rule Number	Count >= .50	Count < 50 and >= .40	Count < .40
1	120	1880	0
2	140	1860	0
3	0	1739	261
4	0	1748	252
5	0	2000	0
6	0	1998	2
7	0	1997	3
8	0	1998	2

Table 5-5

Again, no permutation replications had test statistics exceeding those obtained on the actual data set; as per the test criteria, the null hypothesis was rejected for all eight rules in this sample.

2. Testing of Rules from the Zoo Data Set

The third set of rules tested for significance were from the zoo data set. The rules selected were technically interesting rules induced from the data set which were of high quality based on the confidence fitness function. The performance of the rules tested on the actual data set is presented in Table 5-6. The actual syntax of the rule and the C++ program used to test the rules is included as Appendix E.

Rule Number	# LHS Matched	# Misclassified	Confidence (%)
1	39	8	79.5
2	43	4	90.7
3	17	4	76.5
4	44	12	72.7
5	11	3	72.7
6	27	7	74.1

Table 5-6

The performance of the rule set on 2000 permutation replications is presented in Table 5-7. No scoring statistic from any one permutation replication exceeded the score of any rule on the actual data set. Again, the null hypothesis was rejected for the eight rules. The scoring statistics on the permutation replications exhibited more variation than those on the mushroom data set. It is speculated that this is due to the smaller number of examples in the zoo data set as compared to the mushroom data set (101 vs. 8124) and the larger number of classes (7 vs. 3) present.

Rule Number	Count >= .60	Count< .60 and >= .50	Count < .50 and >= .40	Count <.40 and >= 30	Count < .30 and >= .20	Count < .20
1	2	134	977	814	72	1
2	5	105	852	1003	35	0
3	0	0	2	14	256	1728
4	2	150	971	785	92	0
5	0	0	0	8	74	1918
6	0	0	4	75	829	1092

Table 5-7

B. HYPOTHESIS CONCLUSION

The null hypothesis, that the rules induced by NPSGP perform no differently from rules resulting from random processes, was rejected in all cases. In no cases did test statistics from permutation replications exceed those obtained from the actual data set. Overall, 52,000 permutation replications were generated to test the significance of the 26 rules selected for testing. In the vast majority of cases, the confidence test statistic for the permutation replications were significantly lower than those of the actual test results.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The major objective of this study was to examine the problems and issues associated with statistical hypothesis testing of rules induced from data mining tools. Data mining tools will increasingly play a large part in analyzing the enormous amount of data stored within DOD. Data mining tools such as NPSGP can produce potentially enormous amounts of rules which increase the chance that a rule based on random variation in the data will be accepted as a valid rule. Therefore, it is imperative that some means be available for the testing of induced rules.

Randomization testing represents an attractive testing method for large number of rules based on its freedom from assumptions made in conventional parametric tests. It unquestionably produces accurate results when the methodology is correctly applied. Thus, it is recommended for use in circumstances where the statistical validity of the rule in question must be established to a high degree of accuracy.

All of the twenty six rules tested which were induced from NPSGP outscored the test statistics on the confidence fitness measure produced by two thousand permutation replications. The null hypothesis that NPSGP rules perform no better than rules produced by a random process was rejected for all rules. The empirical results obtained from testing the NPSGP rules showed that NPSGP's clearly produces rules which are better than produced by a random process.

The empirical results also indicated, as expected, that data sets with a larger number of classes and a small number of examples in the data set will experience more variation in the scoring on the confidence fitness measure. Data sets with the above mentioned characteristics will most likely have more chance for random variations in the data which could appear as valid patterns in the data set.

B. RECOMMENDATIONS

More research needs to be conducted on real life data sets to determine if statistical hypothesis testing is warranted for all data sets, especially with data sets with small numbers of classes. If the existing rule induction methods always perform better than can be expected by random, there is no need to conduct the testing.

One of the problems encountered during the study was the computational costs of conducting the randomization testing. For example, the running time of permutation replications for the testing of eight rules in the study took approximately six hours on a Sun SPARC-10 workstation. Additional research could be conducted to optimize methods to conduct the testing so that running times are lessened.

APPENDIX A. MUSHROOM DATA SET

Mushroom Data Set Information

Sources:

(a) Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

(b) Donor to UC- Irvine: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)

Date: 27 April 1987

Past Usage:

1. Schlimmer, J.S. (1987). Concept Acquisition Through Representational Adjustment (Technical Report 87-19). Doctoral disseration, Department of Information and Computer Science, University of California, Irvine.
   --- STAGGER: asymptoted to 95% classification accuracy after reviewing 1000 instances.
2. Iba,W., Wogulis,J., & Langley,P. (1988). Trading off Simplicity and Coverage in Incremental Concept Learning. In Proceedings of the 5th International Conference on Machine Learning, 73-79. Ann Arbor, Michigan: Morgan Kaufmann.
   -- approximately the same results with their HILLARY algorithm

Relevant Information:

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be" for Poisonous Oak and Ivy.

Mushroom Data Set Attributes:

Classes:	Edible, Poisonous, Z
Cap Shape:	Bell, Conical, Convex, Flat, Knobbed, Sunken
Cap Surface:	Fibrous, Grooved, Scaly, Smooth
Cap Color:	Brown, Buff, Cinnamon, Gray, Green, Pink Purple, Red, White, Yellow
Bruises:	True, False
Odor:	Almond, Anise, Creosote, Foul, Musty, Pungent, None
Gill Attachment:	Attached, Descending, Free, Notched
Gill Spacing:	Close (0), Crowded (50), Distant (100)
Gill Size:	Broad, Narrow
Gill Color:	Black, Brown, Buff, Chocolate, Gray, Green, Orange, Pink, Purple, Red, White, Yellow
Stalk Shape:	Enlarging, Tapering
Stalk Root:	Bulbous, Club, Equal, Rhizomorphs, Rooted, Missing
Stalk Surface Above Ring:	Fibrous, Scaly, Silky, Smooth
Stalk Surface Below Ring:	Fibrous, Scaly, Silky, Smooth
Stalk Color Above Ring:	Brown, Buff, Cinnamon, Gray, Orange, Pink, Red
Stalk Color Below Ring:	Brown, Buff, Cinnamon, Gray, Orange, Pink, Red, White, Yellow
Veil Type:	Partial, Universal
Veil Color:	Brown, Orange, White, Yellow
Ring Number:	None (0), One (50), Two(100)

Ring Type:	Cobwebby, Evanescent, Flaring, Large, Pendant, Sheathing, Zone, None
Spore Print Color:	Black, Brown, Buff, Chocolate, Green, Orange, Purple, White, Yellow
Population:	Abundant, Clustered, Numerous, Scattered, Several, Solitary
Habitat:	Grasses, Leaves, Meadows, Paths, Urban, Waste, Woods
Examples:	8124
Distribution of Classes:	Edible (3356), Poisonous (3916), Z (852)

APPENDIX B. ZOO DATA SET

Relevant Information: A simple database containing 17 boolean -valued attributes. The "type" attribute is the class attribute which corresponds to a classification of the animals.

Class Number/ Set of Animals:

1. (41) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, leopard, lynx, mink, mole, mongoose, opossum, oryx, playtpus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal. sealion, squirrel, vampire, vole, wallaby, wolf

2. (20) chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren

3. (5) pitviper, sea snake, slowworm, tortoise, tuatara

4. (13) bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna

5. (4) frog, frog, newt, toad

6. (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp

7. (10) clam, crayfish, octopus, scorpion, seawasp, slug, starfish, worm

Attribute Information:

Number of Attributes: 18 (animal name, 15 boolean attributes, 2 numeric)

Attributes:

1.Animal Name:	Unique for each instance
2.hair:	Boolean
3. feathers:	Boolean
4. eggs:	Boolean
5. milk:	Boolean
6. airborne:	Boolean
7. aquatic:	Boolean
8. predator:	Boolean
9. toothed:	Boolean
10. backbone:	Boolean
11. breathes:	Boolean
12. venomous:	Boolean
13. fins:	Boolean
14. legs:	Numeric (set of values: 0,2,4,6,8)
15. tail:	Boolean
16. domestic:	Boolean
17. catsize:	Boolean
18. type:	Numeric (integer values in range [1..7])

Number of Examples: 101

APPENDIX C. RULES TESTING PROGRAM 1 (MUSHROOM)

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdlib.h>


int rule1_LHS_count = 0;
int rule1_miss_records = 0;
int rule2_LHS_count = 0;
int rule2_miss_records = 0;
int rule3_LHS_count = 0;
int rule3_miss_records = 0;
int rule4_LHS_count = 0;
int rule4_miss_records = 0;
int rule5_LHS_count = 0;
int rule5_miss_records = 0;
int rule6_LHS_count = 0;
int rule6_miss_records = 0;
int rule7_LHS_count = 0;
int rule7_miss_records = 0;
int rule8_LHS_count = 0;
int rule8_miss_records = 0;
int rule9_LHS_count = 0;
int rule9_miss_records = 0;
int rule10_LHS_count = 0;
int rule10_miss_records = 0;
int rule11_LHS_count = 0;
int rule11_miss_records = 0;
int rule12_LHS_count = 0;
int rule12_miss_records = 0;


char rule_name1[] = "Rule 1";
char rule_name2[] = "Rule 2";
char rule_name3[] = "Rule 3";
char rule_name4[] = "Rule 4";
char rule_name5[] = "Rule 5";
char rule_name6[] = "Rule 6";
char rule_name7[] = "Rule 7";
char rule_name8[] = "Rule 8";
char rule_name9[] = "Rule 9";
char rule_name10[] = "Rule 10";
char rule_name11[] = "Rule 11";
char rule_name12[] = "Rule 12";


int records_read=0;
void output(char [], int, int); // function prototype

main()

{
  ifstream inClientFile("mushroom_spa", ios::in);

        if (!inClientFile) {

        cerr << "File could not be opened "<< endl;
        exit(1);  }


        char classification;
        char cap_shape;
        char cap_surface;
        char cap_color;
        char bruises;
        char odor;
        char gill_attach;
        int gill_spacing;
        char gill_size;
        char gill_color;
        char stalk_shape;
        char stalk_root;
        char stalk_surf_abv_ring;
        char stalk_surf_blw_ring;
        char stalk_color_abv_ring;
        char stalk_color_blw_ring;
        char veil_type;
        char veil_color;
```

```
int  ring_number;
char ring_type;
char spore_print_color;
char population;
char habitat;



while (inClientFile >> classification >> cap_shape >> cap_surface >> cap_color
 >> bruises >> odor >> gill_attach >> gill_spacing >> gill_size >> gill_color >>
stalk_shape >> stalk_root >> stalk_surf_abv_ring >> stalk_surf_blw_ring >>
 stalk_color_abv_ring >> stalk_color_blw_ring >> veil_type >> veil_color
   >> ring_number >> ring_type >> spore_print_color >> population >> habitat)


{

        records_read = records_read +1;

        //test rule 1

        if ( stalk_surf_abv_ring == 'k')
           { rule1_LHS_count = rule1_LHS_count + 1;}


        if ( stalk_surf_abv_ring == 'k' && classification != 'p')
            { rule1_miss_records = rule1_miss_records + 1;}


           // test rule 2

        if ( stalk_root == 'c')
           { rule2_LHS_count = rule2_LHS_count +1;}

        if ( stalk_root == 'c' && classification != 'e')
            { rule2_miss_records = rule2_miss_records +1;}


            // test rule 3

        if ( gill_size == 'n')
```

```
    { rule3_LHS_count = rule3_LHS_count +1;}


   if ( gill_size == 'n' && classification != 'p')
     { rule3_miss_records = rule3_miss_records +1;}

             // test rule 4

   if ( stalk_surf_blw_ring != 's')
    { rule4_LHS_count = rule4_LHS_count +1;}

   if ( stalk_surf_blw_ring != 's' && classification != 'p')
      { rule4_miss_records = rule4_miss_records +1;}

            // test rule 5

   if ( cap_shape == 'k')
     { rule5_LHS_count = rule5_LHS_count + 1;}

   if (cap_shape == 'k' && classification != 'p')
    {rule5_miss_records = rule5_miss_records +1;}



            // test rule 6

   if (stalk_root == 'x')
     { rule6_LHS_count = rule6_LHS_count + 1;}

if (stalk_root == 'x' && classification != 'p')
     { rule6_miss_records = rule6_miss_records +1;}


            // test rule 7

   if ( stalk_surf_blw_ring == 'k')
   { rule7_LHS_count = rule7_LHS_count + 1;}


   if (stalk_surf_blw_ring == 'k' && classification != 'p')
      { rule7_miss_records = rule7_miss_records + 1;}
```

```
        // test rule 8

if ( stalk_color_blw_ring == 'n')
   { rule8_LHS_count = rule8_LHS_count + 1;}

if ( stalk_color_blw_ring == 'n' && classification != 'p')
   { rule8_miss_records = rule8_miss_records + 1;}



        // test rule 9

if ( stalk_color_blw_ring == 'p' && !(ring_number >= 9.76 &&
ring_number <= 11.638))
   { rule9_LHS_count = rule9_LHS_count + 1;}


if ( stalk_color_blw_ring == 'p' && !(ring_number >= 9.76 &&
        ring_number <= 11.638) && (classification != 'p'))
   { rule9_miss_records = rule9_miss_records + 1;}


        // test rule 10

if ( gill_size == 'n' && cap_shape == 'x')
   { rule10_LHS_count = rule10_LHS_count + 1;}

if ( gill_size == 'n' && cap_shape == 'x' && classification != 'p')
    { rule10_miss_records = rule10_miss_records + 1;}

    // test rule 11

if ( bruises == 't')
   { rule11_LHS_count = rule11_LHS_count + 1;}

if (bruises == 't' && classification != 'e')
    { rule11_miss_records = rule11_miss_records + 1;}
```

```
            // test rule 12

        if ( gill_size == 'n' && stalk_color_abv_ring == 'w')
           { rule12_LHS_count = rule12_LHS_count + 1;}

        if ( gill_size == 'n' && stalk_color_abv_ring == 'w' &&
                classification != 'p')
                { rule12_miss_records = rule12_miss_records + 1;}


                                          }
           {cout << "Number of records read in: " << records_read << endl;
           cout << endl;
           output( rule_name1, rule1_LHS_count, rule1_miss_records);


           output( rule_name2, rule2_LHS_count, rule2_miss_records);

           output( rule_name3, rule3_LHS_count, rule3_miss_records);
           output( rule_name4, rule4_LHS_count, rule4_miss_records);
           output( rule_name5, rule5_LHS_count, rule5_miss_records);
           output ( rule_name6, rule6_LHS_count, rule6_miss_records);
           output( rule_name7, rule7_LHS_count, rule7_miss_records);
           output ( rule_name8, rule8_LHS_count, rule8_miss_records);
           output ( rule_name9, rule9_LHS_count, rule9_miss_records);
          output ( rule_name10, rule10_LHS_count, rule10_miss_records);
           output (rule_name11, rule11_LHS_count, rule11_miss_records);
           output (rule_name12, rule12_LHS_count, rule12_miss_records);}


    return 0; }


///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    void output( char rule_id[], int LHS_matches , int misclassed_records)

      {    float rule_conf = 1.0 - ((float) misclassed_records/LHS_matches);

           cout << setiosflags(ios::left) <<rule_id << " "<< setw(5) << LHS_matches
           << " " << setw(5) << misclassed_records << " " << setprecision(3) <<
           rule_conf << endl; }
```

APPENDIX D. RULES TESTING PROGRAM 2 (MUSHROOM)

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdlib.h>

int rule1_LHS_count = 0;
int rule1_miss_records = 0;
int rule2_LHS_count = 0;
int rule2_miss_records = 0;
int rule3_LHS_count = 0;
int rule3_miss_records = 0;
int rule4_LHS_count = 0;
int rule4_miss_records = 0;
int rule5_LHS_count = 0;
int rule5_miss_records = 0;
int rule6_LHS_count = 0;
int rule6_miss_records = 0;
int rule7_LHS_count = 0;
int rule7_miss_records = 0;
int rule8_LHS_count = 0;
int rule8_miss_records = 0;


char rule_name1[] = "Rule 1";
char rule_name2[] = "Rule 2";
char rule_name3[] = "Rule 3";
char rule_name4[] = "Rule 4";
char rule_name5[] = "Rule 5";
char rule_name6[] = "Rule 6";
char rule_name7[] = "Rule 7";
char rule_name8[] = "Rule 8";


int records_read=0;
void output(char [], int, int); // function prototype

main()
```

```
{
 ifstream inClientFile("combo_file", ios::in);

        if (!inClientFile) {

                cerr << "File could not be opened "<< endl;
                exit(1);  }



        char classification;
        char cap_shape;
        char cap_surface;
        char cap_color;
        char bruises;
        char odor;
        char gill_attach;
        int gill_spacing;
        char gill_size;
        char gill_color;
        char stalk_shape;
        char stalk_root;
        char stalk_surf_abv_ring;
        char stalk_surf_blw_ring;
        char stalk_color_abv_ring;
        char stalk_color_blw_ring;
        char veil_type;
        char veil_color;
        int  ring_number;
        char ring_type;
        char spore_print_color;
        char population;
        char habitat;


        while (inClientFile >> classification >> cap_shape >> cap_surface >> cap_color
         >> bruises >> odor >> gill_attach >> gill_spacing >> gill_size >> gill_color >>
        stalk_shape >> stalk_root >> stalk_surf_abv_ring >> stalk_surf_blw_ring >>
         stalk_color_abv_ring >> stalk_color_blw_ring >> veil_type >> veil_color
           >> ring_number >> ring_type >> spore_print_color >> population >> habitat)
```

```
{
        records_read = records_read +1;

        // test rule 1

        if ( cap_color == 'y')
           { rule1_LHS_count = rule1_LHS_count + 1;}


        if ( cap_color == 'y' && classification != 'p')
          { rule1_miss_records = rule1_miss_records + 1;}


           // test rule 2

        if ( stalk_color_abv_ring == 'p')
       { rule2_LHS_count = rule2_LHS_count +1;}

        if ( stalk_color_abv_ring == 'p' && classification != 'p')
          { rule2_miss_records = rule2_miss_records +1;}



                  // test rule 3
        if ( gill_color == 'w')
          { rule3_LHS_count = rule3_LHS_count +1;}

        if ( gill_color == 'w' && classification != 'e')
          { rule3_miss_records = rule3_miss_records +1;}

         // test rule 4

        if ( gill_color == 'n')
          { rule4_LHS_count = rule4_LHS_count +1;}

        if ( gill_color == 'n' && classification != 'e')
          { rule4_miss_records = rule4_miss_records +1;}

                // test rule 5

        if ( gill_size == 'b')
```

```
      { rule5_LHS_count = rule5_LHS_count + 1;}

    if ( gill_size == 'b' && classification != 'e')
  {rule5_miss_records = rule5_miss_records +1;}

            // test rule 6

    if ( stalk_root != 'e')
       { rule6_LHS_count = rule6_LHS_count + 1;}

if (stalk_root != 'e' && classification != 'p')
      { rule6_miss_records = rule6_miss_records +1;}

            // test rule 7

    if ( stalk_surf_blw_ring == 's')
    { rule7_LHS_count = rule7_LHS_count + 1;}

    if  ( stalk_surf_blw_ring == 's' && classification != 'e')
        { rule7_miss_records = rule7_miss_records + 1;}

            // test rule 8

    if ( (stalk_color_blw_ring == 'p') || (gill_size == 'b'))
       { rule8_LHS_count = rule8_LHS_count + 1;}

    if ( ((stalk_color_blw_ring == 'p') || (gill_size == 'b')) &&
            (classification != 'e') )
       { rule8_miss_records = rule8_miss_records + 1;} }
                    // end of testing conditions block


      { cout << "Number of records read in: " << records_read << endl;
       output( rule_name1, rule1_LHS_count, rule1_miss_records);
       output( rule_name2, rule2_LHS_count, rule2_miss_records);
       output( rule_name3, rule3_LHS_count, rule3_miss_records);
       output( rule_name4, rule4_LHS_count, rule4_miss_records);
       output( rule_name5, rule5_LHS_count, rule5_miss_records);
       output ( rule_name6, rule6_LHS_count, rule6_miss_records);
       output( rule_name7, rule7_LHS_count, rule7_miss_records);
       output ( rule_name8, rule8_LHS_count, rule8_miss_records);  }
```

```
return 0; }


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void output(char rule_id[], int LHS_matches , int misclassed_records)

    {    float rule_conf = 1.0 - ((float) misclassed_records/LHS_matches);

         cout << setiosflags(ios::left) <<rule_id << " "<< setw(5) <<
         LHS_matches << " " << setw(5) << misclassed_records << " " <<
         setprecision(3) << rule_conf << endl; }
```

APPENDIX E. RULES TESTING PROGRAM (ZOO)

```
#include <iostream.h>

#include <fstream.h>#include <iomanip.h>

#include <stdlib.h>

int rule1_LHS_count = 0;
int rule1_miss_records = 0;
int rule2_LHS_count = 0;
int rule2_miss_records = 0;
int rule3_LHS_count = 0;
int rule3_miss_records = 0;
int rule4_LHS_count = 0;
int rule4_miss_records = 0;
int rule5_LHS_count = 0;
int rule5_miss_records = 0;
int rule6_LHS_count = 0;
int rule6_miss_records = 0;
int rule7_LHS_count = 0;
int rule7_miss_records = 0;
int rule8_LHS_count = 0;
int rule8_miss_records = 0;
int rule9_LHS_count = 0;
int rule9_miss_records = 0;
int rule10_LHS_count = 0;
int rule10_miss_records = 0;
int rule11_LHS_count = 0;
int rule11_miss_records = 0;
int rule12_LHS_count = 0;
int rule12_miss_records = 0;

char rule_name1[] = "Rule 1";
char rule_name2[] = "Rule 2";
char rule_name3[] = "Rule 3";
char rule_name4[] = "Rule 4";
char rule_name5[] = "Rule 5";
char rule_name6[] = "Rule 6";
char rule_name7[] = "Rule 7";
char rule_name8[] = "Rule 8";
char rule_name9[] = "Rule 9";
char rule_name10[] = "Rule 10";
char rule_name11[] = "Rule 11";
char rule_name12[] = "Rule 12";

int records_read=0;
void output(char [], int, int); // function prototype

main()


{
  ifstream inClientFile("gpzoo.tab", ios::in);

        if (!inClientFile) {

        cerr << "File could not be opened "<< endl;
        exit(1);  }


        char name[10];
        int hair;
        int feathers;
        int eggs;
        int milk;
        int airborne;
        int aquatic;
        int predator;
        int toothed;
        int backbone;
        int breathes;
        int venomous;
        int fins;
        int legs;
        int tail;
        int domestic;
        int catsize;
        int classification;
```

```
while (inClientFile >> name >> hair >> feathers >> eggs
 >> milk >> airborne >> aquatic >> predator >> toothed >> backbone >>
breathes >> venomous >> fins >> legs >> tail >> domestic >> catsize >>
classification)

        {
                records_read = records_read +1;


                //test rule 1

        if  (legs >= 3.6 && legs <= 5.9)
           { rule1_LHS_count = rule1_LHS_count + 1;}


        if ( (legs >= 3.6 && legs <= 5.9) && (classification != 1))
          { rule1_miss_records = rule1_miss_records + 1;}


           // test rule 2

        if ( hair  == 1)
        { rule2_LHS_count = rule2_LHS_count + 1;}

        if ( hair  == 1 && classification != 1 )
            { rule2_miss_records = rule2_miss_records +1;}


         // test rule 3

     if ( fins == 1)
        { rule3_LHS_count = rule3_LHS_count +1;}


     if ( fins == 1 && classification != 4)
          { rule3_miss_records = rule3_miss_records +1;}

          // test rule 4
```

```
if ( catsize == 1)
  { rule4_LHS_count = rule4_LHS_count +1;}

if ( catsize == 1 && classification != 1)
    { rule4_miss_records = rule4_miss_records +1;}

        // test rule 5

if ( legs >= 4.3 && legs <= 7.0)
{ rule5_LHS_count = rule5_LHS_count + 1;}

if ( (legs >= 4.3 && legs <= 7.0) && (classification != 6))
  {rule5_miss_records = rule5_miss_records +1;}



        // test rule 6

if ( legs >= 1.2 && legs <= 3.9)
   { rule6_LHS_count = rule6_LHS_count + 1;}

if (( legs >= 1.2  && legs <= 3.9) &&  (classification != 2))
    { rule6_miss_records = rule6_miss_records +1;}



        // test rule 7

if ( catsize == 0)
{ rule7_LHS_count = rule7_LHS_count + 1;}

if ( ( catsize == 0 ) && (classification != 1) )
    { rule7_miss_records = rule7_miss_records + 1;}



        // test rule 8

if ( eggs == 0)
   { rule8_LHS_count = rule8_LHS_count + 1;}
```

```
        if ( ( eggs == 0 ) && (classification != 1))
          { rule8_miss_records = rule8_miss_records + 1;}




                                                  }
          cout << "Number of records read in: " << records_read << endl;

    output( rule_name1, rule1_LHS_count, rule1_miss_records);
    output( rule_name2, rule2_LHS_count, rule2_miss_records);
    output( rule_name3, rule3_LHS_count, rule3_miss_records);
    output( rule_name4, rule4_LHS_count, rule4_miss_records);
    output( rule_name5, rule5_LHS_count, rule5_miss_records);
    output ( rule_name6, rule6_LHS_count, rule6_miss_records);
    output( rule_name7, rule7_LHS_count, rule7_miss_records);
    output ( rule_name8, rule8_LHS_count, rule8_miss_records);}

  return 0; }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

  void output( char rule_id[], int LHS_matches , int misclassed_records)

     {    float rule_conf = 1.0 - ((float) misclassed_records/LHS_matches);

          cout << setiosflags(ios::left) <<rule_id << " "<< setw(5) << LHS_matches
          << " " << setw(5) << misclassed_records << " " << setprecision(3) <<
          rule_conf << endl; }
```

LIST OF REFERENCES

[Citation by Number:]

1. Frawley, William J., Gregory Piatetsky-Shapiro and Christopher J. Matheus "Knowledge Discovery in Databases: An Overview." in *Knowledge Discovery in Databases,* eds Gregory Piatetsky-Shapiro and William J. Frawley, AAAI Press, Menlo Park, CA 1991.

2. Simon, Herbert, A., "Why Should Machines Learn?" in *Machine Learning, An Artificial Intelligence Approach,* eds. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Morgan Kaufman Inc., Los Altos, CA 1983.

3. Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., "An Overview of Machine Learning" in *Machine Learning, An Artificial Intelligence Approach,* eds. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Morgan Kaufman Inc., Los Altos, CA 1983.

4. Kloesgen, W., and Zytkow, J., "Machine Discovery Terminology", Working Paper, 1994.

5. Goldberg, David, E. *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley Inc., 1989.

6. Koza, John, R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

7. Wuthrich, Beat, Course Notes, Comp631, Knowledge Discovery in Databases, Hong Kong Institute of Science and Technology, 1994.

8. Piatetsky-Shapiro, Gregory "Discovery, Analysis, and Presentation of Strong Rules" in *Knowledge Discovery in Databases,* eds Gregory Piatetsky-Shapiro and William J. Frawley, AAAI Press, Menlo Park, CA 1991.

9. Smyth, Padhraic and Rodney M. Goldman, "Rule Induction Using Information Theory" in *Knowledge Discovery in Databases,* eds Gregory Piatetsky-Shapiro and William J. Frawley, AAAI Press, Menlo Park, CA 1991.

10. Jones, D.S., *Elementary Information Theory,* Oxford University Press, 1979.

11. Major, John A. and Magano, John J., *"Selecting Among Rules Induced from a Hurricane Database.",* Knowledge Discovery in Databases Workshop, 1993.

12. Sachs, Lothar, *Applied Statistics, A Handbook of Techniques,* Springer-Verlag, New York 1982.

13. Jensen, David "Induction with Randomization Testing: Decision-Oriented Analysis of Large Data Sets", Sever Institute of Technology, Washington University, St. Louis Missouri 1992.

14. Jensen, David, "Concept Reliability in Machine Learning." *Proceedings of the Second Midwest AI and Cognitive Science Society Conference*, eds. Dinsmore, John and Tim Koschmann, 1990.

15. Selvin, Hanan C., and Alan Stuart, " Data Dredging Procedures in Survey Analysis." *American Statistician,* Vol 20, June 1966.

16. Efron, Bradley, and Rob Tibshirani, *An Introduction to the Bootstrap*, Chapman and Hall, New York 1993.

17. Edgington, Eugene S., *Randomization Tests*, Marcel Dekker, New York 1987.

18. *Audubon Society Field Guide to North American Mushrooms,* G. H. Lincoff (Pres.), Alfred A. Knopf, New York 1981.

19. Bunn, Frank J., and Elizabeth Walters, "Knowledge Quality Functions for Rule Discovery", Master's Thesis, Naval Postgraduate School, 1994.

20. Whitten, Brian D., "Comparative Analysis of Techniques for Classification with Respect to Rule Interestingness.", Master's Thesis, Naval Postgraduate School, 1994.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. Balasubramanium Ramesh, Code SM/RA Department of Systems Management Naval Postgraduate School Monterey, CA 93943-5000	2
4.	Dr. William J. Haga, Code SM/HG Department of Systems Management Naval Postgraduate School Monterey, CA 93943-5000	2
5.	LCDR Eric D. Berry Code 92 Navy Fleet Material Support Office 5450 Carlisle Pike P.O. Box 2010 Mechanicsburg, PA 17055-0787	2